Image Watermarking dengan Menggunakan Message Digest

Dimpos Arie Ginarta Sitorus / 13513083
Sekolah Teknik Elektro dan Informatika
Instiut Teknologi Bandung
Bandung, Indonesia
dimpossitorus@gmail.com

***Abstrak*—*Image Watermarking* adalah salah satu cara untuk melindungi keaslian gambar. Dalam makalah ini diajukan teknik baru untuk image watermarking, yaitu dengan menyisipkan *digital signature* pada gambar.**

***Kata Kunci*—*Watermarking, Image Watermarking, Digital Signature, Hash Function*.**

I. PENDAHULUAN

Banyak cara dilakukan orang untuk melindungi keaslian dari dokumen/gambar miliknya. Pada awalnya dilakukan dengan cara memberikan label *copyright* pada gambar tersebut. Kelebihan cara tersebut adalah cukup mudah untuk dilakukan. Kekurangannya adalah cara ini tidak cukup efektif melindungi hak cipta gambar, karena label tersebut dapat dipotong atau dibuang dengan manipulasi tertentu.

Kemudian berkembang *digital watermarking*. Khusus pada gambar disebut *image watermarking*. *Image watermarking* adalah salah satu cara melindungi keaslian gambar. Banyak cara membuat watermark pada gambar,

II. DASAR TEORI

A. *Image Watermarking*

Image watermarking adalah penyisipan informasi (watermark) oleh pemilik gambar untuk tujuan melindungi kepemilikan, copyright atau menjaga keaslian konten. Watermark bisa dalam bentuk teks, gambar logo, audo, data *binary* (+1/-1), barisan bilangan riil, dll. Dalam prosesnya diusahakan penyisipan watermark tersebut tidak merusak kualitas citra. Terdapat dua jenis image watermarking, yaitu *Fragile Watermarking* dan *Robust Watermaking*.

Fragile watermarking bertujuan untuk menjaga integritas/orisinalitas dari media digital ketika akan dikirim ke orang/tujuan tertentu. Watermark akan rusak jika terjadi perubahan atau dilakukan manipulasi (*common digital processing)* pada gambar.

Robust Watermarking bertujuan untuk menyisipkan label kepemilikan pada media digital. Watermark akan tetap kokoh (*robust)* jika terjadi perubahan atau dilakukan manipulasi (*common digital processing*) pada gambar.

Image watermarking tidak boleh merusak citra gambar secara visual.

B. *Fungsi Hash*

Fungsi hash menerima input masukan *string/array of byte* yang panjangnya sembarang dan kemudian mentransformasikannya menjadi sebuah string yang panjangnya tetap dan pada umummnya jauh lebih kecil dari string input.

Pada umumnya fungsi hash adalah fungsi satu arah (*one-way function)*. Hasil dari fungsi hash, disebut *message digest*, tidak dapat dikembalikan menjadi pesan semula.

Terdapat beberapa aplikasi dari fungsi hash, yaitu :

1. Menjaga integritas data

   Fungsi hash sangat sensitif terhadap perubahan data. Jika data berubah satu bit, nilai hash akan berubah sangat signifikan. Hal ini dapat digunakan untuk melakukan verifikasi data dengan sumber aslinya.

2. Menghemat waktu pengiriman

   Sebagai contoh untuk melakukan verifikasi salinan arsip, cukup kirim *message digest* dari dokumen tersebut. Waktu pengiriman *message digest* akan lebih cepat daripada waktu pengiriman dokumen asli. Dan jika *message digest* dokumen asli dan salinan arsip sama, makan salinan arsip tersebut masih sama dengan dokumen asli.

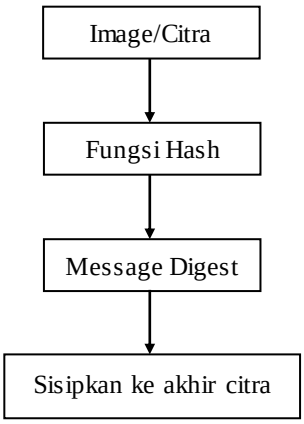
3. Menormalkan panjang data yang beraneka ragam

   Contoh pada password yang panjangnya bebas, dan disimpan di basis data. Untuk menyamakan panjang *field password* di basisdata, *password* tersebut disimpan dalam bentuk nilai hash.

III. RANCANGAN DAN PSEUDOCODE

A. *Rancangan*

Gambar dikonversi menjadi *stream of byte*. Lalu kemudian menjadi input untuk fungsi hash. Kemudian dihasilkan *message digest* dan disisipkan ke dalam citra



IV. IMPLEMENTASI DAN UJICOBA

Implementasi dan ujicoba menggunakan fungsi Hash SHA-1.

A. *Implementasi*

Digest.java

```
package secureimage;

public class Digest {
        int j, temp;
        int A, B, C, D, E;
        int[] H = {0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476, 0xC3D2E1F0};
        int F;

        public Digest() {

        }

        public String sha1(String text) {
                return sha1(text.getBytes());
        }

        public String sha1(byte[] dataIn) {
                byte[] paddedData = padMessage(dataIn);
                int[] H = {0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476, 0xC3D2E1F0};
                int[] K = {0x5A827999, 0x6ED9EBA1, 0x8F1BBCDC, 0xCA62C1D6};

                if (paddedData.length % 64 != 0) {
                        System.out.println("Invalid padded data length.");
                        return "";
                }

                int passesReq = paddedData.length / 64;
                byte[] work = new byte[64];
```

```
                for (int passCntr = 0; passCntr < passesReq; passCntr++) {
                        System.arraycopy(paddedData, 64 * passCntr, work, 0, 64);
                        processBlock(work, H, K);
                }

                return intArrayToHexStr(H);
        }

        private byte[] padMessage(byte[] data) {
                int origLength = data.length;
                int tailLength = origLength % 64;
                int padLength = 0;
                if ((64 - tailLength >= 9)) {
                        padLength = 64 - tailLength;
                }
                else {
                        padLength = 128 - tailLength;
                }

                byte[] thePad = new byte[padLength];
                thePad[0] = (byte) 0x80;
                long lengthInBits = origLength * 8;

                for (int cnt = 0; cnt < 8; cnt++) {
                        thePad[thePad.length - 1 - cnt] = (byte) ((lengthInBits >> (8 * cnt)) &
0x00000000000000FF);
                }

                byte[] output = new byte[origLength + padLength];

                System.arraycopy(data, 0, output, 0, origLength);
                System.arraycopy(thePad, 0, output, origLength, thePad.length);

                return output;
        }

        private void processBlock(byte[] work, int H[], int K[]) {

                int[] W = new int[80];
                for (int outer = 0; outer < 16; outer++) {
                        int temp = 0;
                        for (int inner = 0; inner < 4; inner++) {
                                temp = (work[outer * 4 + inner] & 0x000000FF) << (24 - inner * 8);
                                W[outer] = W[outer] | temp;
                        }
                }

                for (int j = 16; j < 80; j++) {
                        W[j] = rotateLeft(W[j - 3] ^ W[j - 8] ^ W[j - 14] ^ W[j - 16], 1);
                }

                A = H[0];
                B = H[1];
                C = H[2];
                D = H[3];
                E = H[4];

                for (int j = 0; j < 20; j++) {
                        F = (B & C) | ((~B) & D);
                        temp = rotateLeft(A, 5) + F + E + K[0] + W[j];
                        E = D;
                        D = C;
                        C = rotateLeft(B, 30);
                        B = A;
                        A = temp;
                }

                for (int j = 20; j < 40; j++) {
                        F = B ^ C ^ D;
                        temp = rotateLeft(A, 5) + F + E + K[1] + W[j];
                        E = D;
                        D = C;
                        C = rotateLeft(B, 30);
```

```
                        B = A;
                        A = temp;
                }

                for (int j = 40; j < 60; j++) {
                        F = (B & C) | (B & D) | (C & D);
                        temp = rotateLeft(A, 5) + F + E + K[2] + W[j];
                        E = D;
                        D = C;
                        C = rotateLeft(B, 30);
                        B = A;
                        A = temp;
                }

                for (int j = 60; j < 80; j++) {
                        F = B ^ C ^ D;
                        temp = rotateLeft(A, 5) + F + E + K[3] + W[j];
                        E = D;
                        D = C;
                        C = rotateLeft(B, 30);
                        B = A;
                        A = temp;
                }

                H[0] += A;
                H[1] += B;
                H[2] += C;
                H[3] += D;
                H[4] += E;
        }

        final int rotateLeft(int value, int bits) {
                int q = (value << bits) | (value >>> (32 - bits));
                return q;
        }

        private String intArrayToHexStr(int[] data) {
                String output = "";
                String tempStr = "";
                int tempInt = 0;
                for (int cnt = 0; cnt < data.length; cnt++) {
                        tempInt = data[cnt];
                        tempStr = Integer.toHexString(tempInt);
                        if (tempStr.length() == 1) {
                                tempStr = "0000000" + tempStr;
                        } else if (tempStr.length() == 2) {
                                tempStr = "000000" + tempStr;
                        } else if (tempStr.length() == 3) {
                                tempStr = "00000" + tempStr;
                        } else if (tempStr.length() == 4) {
                                tempStr = "0000" + tempStr;
                        } else if (tempStr.length() == 5) {
                                tempStr = "000" + tempStr;
                        } else if (tempStr.length() == 6) {
                                tempStr = "00" + tempStr;
                        } else if (tempStr.length() == 7) {
                                tempStr = "0" + tempStr;
                        }
                        output = output + tempStr;
                }
                return output;
        }
}
```

SecureImage.java

```
package secureimage;


import java.awt.image.BufferedImage;

import java.io.ByteArrayInputStream;
```

```
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.net.URISyntaxException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.IIOImage;
import javax.imageio.ImageIO;
import javax.imageio.ImageWriteParam;
import javax.imageio.ImageWriter;
import javax.imageio.stream.FileImageOutputStream;

public class SecureImage {

    public SecureImage() {
    }

    public byte[] getImageByte(String filename) {
     ByteArrayOutputStream stream = new ByteArrayOutputStream();
     BufferedImage image;
     byte[] imageByte = null;
     try {
         image = ImageIO.read(new File(getClass().getResource("/image/"+filename).toURI()));
         ImageIO.write(image, "jpg", stream);
         stream.flush();
         imageByte = stream.toByteArray();
         stream.close();
     }
     catch (IOException e) {
         e.printStackTrace();
     } catch (URISyntaxException ex) {
         Logger.getLogger(SecureImage.class.getName()).log(Level.SEVERE, null, ex);
     }
     return imageByte;
    }

    public String getHash(String filename) {
     return (new Digest().sha1(getImageByte(filename)));
    }

    public String getHash(byte[] fileByte) {
     return (new Digest().sha1(fileByte));
    }
```

```
public void signatureImage(String filename) {
 BufferedImage image;
 byte[] imageByte = getImageByte(filename);

 String hash = getHash(filename);
 byte[] hashByte = hash.getBytes();
 System.out.println(hash);
 System.out.println(imageByte.length);
 System.out.println(hashByte.length);

 byte[] signatureByte = new byte[imageByte.length + hashByte.length];
 System.arraycopy(imageByte, 0, signatureByte, 0, imageByte.length);
 System.arraycopy(hashByte, 0, signatureByte, imageByte.length, hashByte.length);
 System.out.println(signatureByte.length);

 try {
     image = ImageIO.read(new ByteArrayInputStream(signatureByte));
     ImageWriter writer = ImageIO.getImageWritersByFormatName("jpg").next();
     ImageWriteParam param = writer.getDefaultWriteParam();
     param.setCompressionMode(ImageWriteParam.MODE_COPY_FROM_METADATA);
     String urlFile = "D:/Dimpos/NetBeansProjects/SecureImage/src/image/signature-"+filename;
     File output = new File(urlFile);
     writer.setOutput(new FileImageOutputStream(output));
     writer.write(null, new IIOImage(image, null,null), param);
 } catch (IOException ex) {
     Logger.getLogger(SecureImage.class.getName()).log(Level.SEVERE, null, ex);
 }
}

public boolean checkSignature(String filename) {
 byte[] signatureByte = getImageByte(filename);
 byte[] hashByte = new byte[40];
 System.out.println(signatureByte.length);
 System.out.println(hashByte.length);
 String hash = "";
 System.arraycopy(signatureByte, signatureByte.length-40, hashByte, 0, 40);
 try {
     hash = new String (hashByte, "UTF-8");
     System.out.println(hash);
 } catch (UnsupportedEncodingException ex) {
     Logger.getLogger(SecureImage.class.getName()).log(Level.SEVERE, null, ex);
 }
```

```
        byte[] imageByte = new byte[signatureByte.length-40];
        System.out.println(imageByte.length);
        System.arraycopy(signatureByte, 0, imageByte, 0, signatureByte.length-40);
        String hashSignatureImage = getHash(imageByte);
        System.out.println(hashSignatureImage);
        return hash.equals(hashSignatureImage);
    }
```

Main.java

```
package secureimage;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

/**
 *
 * @author B2-12
 */
public class Main {
     /**
      * @param args the command line arguments
      */
     public static void main(String[] args) {
      SecureImage secureImage = new SecureImage();
      secureImage.signatureImage("process-ariana.jpg");
      System.out.println(secureImage.checkSignature("signature-process-ariana.jpg"));
      }
}
```

B. Ujicoba

Pada implementasi diuji coba untuk melakukan signature pada sebuah gambar dengan nama “ariana.jpg” didapatkan hasil log seperti berikut :

Hash : f550b78ec0efd79682b086c89398449769d5ae40

Image Size : 41856

Signatured Image Size : 41896

Check Signatured ImageSize : 41808

Check Signatured Hash : ~(-
j/j U E h7^ _J(??

Check Signatured Image Hash :
0757a222c5b398dc423b16edd7b14c90c3b95bd0

Signature Check :false

Ujicoba untuk file ariana2.jpg :

Hash : fc6aa47d2c460924315196e1ffc164d68a741605

Image Size : 173864

Signatured Image Size : 173904

Check Signatured ImageSize : 173857

Check Signatured Hash : (QE

Check Signatured Image Hash :
7b445863b5798c9c11f751cabbf2f73732afe3eb

Signature Check :false

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Algoritma ini berfungsi dengan baik. Dalam proses penyimpanan gambar yang telah diberikan watermark signature, terjadi kompresi, yang mana kompresi mengubah bit gambar, sehingga hasil cek hash image dan signature berbeda.

B. Saran

Algoritma ini dapat dikembangkan menjadi algoritma *robust watermarking*.

REFERENSI

[1] Schneier, Bruce. “Applied Cryptography”, 2nd ed. John Wiley & Sons, 1994.

[2] Slide kuliah “Digital Watermarking”, IF4020 Kriptografi

[3] Slide kuliah “Fungsi Hash”, IF4020 Kriptografi

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi

Bandung, 18 Desember 2016

Dimpos Sitorus